東京大学大学院新領域創成科学研究科
社会文化環境学専攻

2018 年度
修　士　論　文

都市における人の移動性に対して VAE に基づく生成モデルの検討
A Variational Autoencoder Based Generative Model of Urban Human Mobility

2018 年 7 月 13 日提出
指導教員　柴崎　亮介　教授

黄　豆
Huang, Dou

Abstract

Recently, big and heterogeneous human mobility data inspires many revolutionary ideas of implementing machine learning algorithms for solving some traditional social issues, such as zone regulation, air pollution, and disaster evacuation el at.. However, incomplete datasets were provided owing to both the concerns of invasion of privacy and some technique issues in many practical applications, which leads to some limitations of the utility of collected data. Inspired by the generative model used for reconstructing images in image processing domain, we want to build a generative model which can tackle the human mobility data. Variational Autoencoder (VAE), which uses a well-constructed latent space to capture salient features of the training data, shows a significant excellent performance in not only image processing, but also Natural Language Processing domain. By combining VAE and sequence-to-sequence (seq2seq) model, a Sequential Variational Autoencoder (SVAE) is built for the task of human mobility reconstruction. It is the first time that this kind of SVAE model is implemented for solving the issues about human mobility reconstruction. We use navigation GPS data of selected greater Tokyo area to evaluate the performance of the SVAE model. Experimental results demonstrate that the SVAE model can efficiently capture the salient features of human mobility data and generate more reasonable trajectories. That indicates the applicability of the SVAE to real-world urban computing problems.

Keywords: big data, urban computing, GPS trajectory, generative model

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisors Prof. Shibasaki, Prof. Song and Dr. Kusakabe for the continuous support of my study and research. Prof. Shibasaki always gives me a lot of valuable information and advices for my research. Prof. Song's guidance helped me in all the time of research and presentations throughout my master's course study in the IPUC group. My sincere thanks also goes to Dr. Kusakabe for his patient guidance of my master thesis. Besides, I would like to thank my labmates in IPUC group: Zipei Fan, Quanjun Chen, Satoshi Miyazawa, Renhe Jiang, Tianqi Xia, Shuzhe Huang. They support me for solving the technique problems in the research of my master thesis. Last but not the least, I would like to thank Researcher Kanasugi for data infrastructure and Researcher Ueyama for visualization tool.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many big cities have grown thanks for the rapid urbanization progress, which have modernized many people's lives but also engendered big challenges.[1] Years ago, solving this kind of chanllenges seems to be impossible because of the complex and dynamic settings of cities. Nowadays, some impressive methods of locational datasets collection have shown an opportunity for the human mobility applications. For example, human mobility in a city which occurs during some rare events like earthquake was recorded, then how can we use this data to evaluate the situation if the earthquake happened in another city. Although the usage of those kinds of datasets, which owned by enterprises or government, can give us opportunities to some potential applications, they have some limitations in two-fold: 1) it has the risk of an invasion of privacy in some cases if used directly; 2) it will contain some bias or the sampling rate is low.

1.1 Problem definition

Privacy is a very complex topic. To prevent some risks of invasion of privacy of mobile device users, although many locational data is collected, the owner of such kind of data is not willing to provide the data for researchers or other research institutes, which leads to a limitation of the usage of this kind of locational data. There is a trade-off between the implement of collected locational data and the concerns about invasion of privacy of users, which is often that even the owner of the data is willing to provide the data for some research purpose, they are not going to provide the whole data, instead, a very small part of the data will be

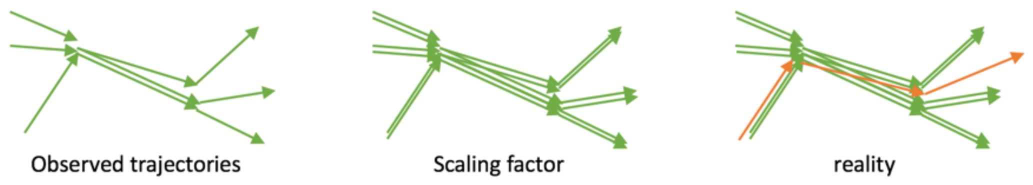


Figure 1.1: Example of scaling factor.

provided, such as 1% of the entire data set. Despite the privacy concerns about protecting the privacy of users, there are some other problems which can also lead to the low sampling rate of the collected data. An example of such situation is that fishery data in the world. This kind of fishery data is an open data, but we still cannot get the data reflect the real trajectory patterns of all fishing boats since some of small fishing boats lack efficient device to record their trajectories and thus cannot be obtained. Both privacy concerns of mobile device users and the lack of techniques of collection method in some cases will lead to the difficulty to obtain the human mobility data to reflect the real trajectory patterns in real situations. There are many research and implement based on the human mobility data, for instance, human mobility prediction. These applications usually need to use previous steps of trajectories to predict the human mobility in the future. We are not talking about the accuracy or performance of such methods, what we concern is that if we cannot get the data which can reflect the real situation of human mobility in a target area, it is difficult to predict the future human mobility in a proper way.

1.2 Scaling factor

For some human mobility prediction problem which aims to predict the human mobility in a target area, it is necessary to know the real situation about the current human mobility. However, in reality, the provided data cannot reflect that real situation if the data only contains 1% of entire population in real world. To tackle this kind of problem, we can develop a scaling factor for each trajectory sample by combining some information, such as population density, from other data set.

Figure 1.1 shows an example of scaling factor for human mobility trajectories. In left panel, it shows observed trajectories; middle one shows trajectories after scaling factor applied; right one shows real situation of trajectories. The scaling factor can add more trajectories based on observed trajectories to make the data approximate the real situation of human mobility in a target area. However, its limitations are also obvious. It can only add some trajectories based on the existed observations, thus it is surely lack of diversity as different people is assumed to behave in somehow different even though some of they may be in similar situation. As shown in the most right penal of the above figure, in reality, there might be some potential trajectories which didn't observed. It is more reasonable to achieve a diversity of trajectories when reconstructing the real human mobility patterns.

1.3 Generative model

In general, a generative model is a model of the conditional probability of the observable X, given a target y, symbolically, $P(X|Y = y)$.[2] It can be used to generate random outcomes, either of an observation and target (x, y), or of an observation x given a target value y. A generative model is not designed for the transportation planning and applications directly, but we can use this kind of model to improve the existing datasets to match the requirement of implementation of other applications. This kind of model can solve the limitations of aforementioned scaling factor. First of all, a generative model can learn a low dimensional feature space which can infer the travelers' pattern from the complex redundant collected locational datasets. Then, we can utilize the learned feature space to transportation planning and applications. And if necessary, we can resample from the learned low dimensional feature space to generate a fake dataset that has the similar pattern with the real dataset for further use. There are two reasons for generating fake datasets: 1) using generated fake datasets can avoid the risk of invasion of customers' privacy; 2) obtain enough data samples if the dataset is too small to be used. Therefore, the problem that how to build a generative model that can capture the features from real human mobility trajectories is a very interesting topic. Nowadays, many deep learning methods have been investigated on image processing, natural language processing and human mobility prediction, et al. based on basic neural networks, many different neural networks have been proposed to solve different problems in various domains. Although, different deep learning frameworks were proposed for solving the problems lying on very different implementations, we can still be inspired from those deep learning frameworks. Variational Autoencoders (VAE)[3] are originally proposed for image processing, and many applications using variational autoecncoders achieved a very good performance. However, owing to the structure of variational autoencoders, it can only be implemented in applications using non-sequential data. To tackle problems of human mobility, which is a kind of sequential data, we need use Recurrent Neural Networks (RNN) to build our model. Since vanilla RNNs have difficulties on long length sequence training owing to vanishing gradient problem, Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) have been designed and widely used when coming to the long length sequence problem.

1.4 Contributions

In this paper, we combine these two frameworks and build a Sequential Variational Autoencoders (SVAE) model for tackling trajectories of human mobility. In our limited knowledge, it is the first time to implement the SVAE for trajectories of human mobility. In the model architecture we focus on learning the hidden space which using multivariate Gaussian distribution to approximate the real posterior distribution of the real human mobility. Then we can resample from the learned distribution, and using a learned decoder to reconstruct the trajectories of human mobility. Our contributions are as follows:
1) Controlled generation. Our architecture allows us to generate more trajectories which follows the distribution of the learned human mobility pattern. We can control the number of trajectories we want to generate.
2) Diversity. We can obtain some reasonable trajectories, which are not contained in the original data set, to achieve diversity.
3) We use some metrics to quantitatively evaluate the performance of SVAE model for trajectories of human mobility.
We use real navigation GPS data to conduct the experiment. The data we used is locational data contains trajectories of human mobility of entire Japan. Our paper is structured as follows. In section 2, we introduce some related works. In section 3, we present the framework of SVAE model. In section 4, experiment design is presented. Results of experiment are shown in section 5. We make a brief discussion in section 6. Finally conclusion and further study were given in section 7.

Chapter 2

Related Works

2.1 Urban computing using human mobility data

A definition of Urban Computing is given by Y. Zheng[1]: it is a process of tackling the major issues which cities face using big and heterogeneous data collected by a diversity of sources in urban areas.

Techniques of data collection have been improved rapidly, which lead to some revolutionary ideas of implementing machine learning algorithms for solving some traditional social issues, such as zone regulation[4], air pollution[5], disaster evacuation[6] el at., since collected big and heterogeneous data makes tasks which are nearly impossible years ago become possible.

Recently, there are many researches conducted on human mobility data, such as mobile phone GPS log data, taxi GPS data and navigation GPS data. These kinds of researches often related with building intelligent city system.

R, Jiang[7] introduce a framework of predicting multiple steps of future trajectories of human mobility. Their method is a Regions-of-Interest (ROIs) based modeling, which is convinced to be an improvement of traditional grid based modeling. Also, they use a multiple to multiple training strategy to achieve the goal of predicting the multiple steps of future movement.

CityMomentum[8] is another work related to human mobility prediction. However, the goal of building CityMomentum system is not to predict future trajectories of human mobility using previous historical trajectories, but to transfer the information obtained in one city to another city. It is also a very interesting work which answers the question about how to make use of data collected in one city to guide the development of another city.

Detecting flawed urban planning using the GPS trajectories of taxicabs[9] is one of the most significant example of urban computing for city planning. Their work can detect the regions with salient traffic problems and the linking structure as well as correlation among them.

Furthermore, some other researches about simulating human mobility when disasters occur and predict their mobility in an emergency have been also conducted[10–14]. Their works are extremely important since understanding and modeling people's mobility is a crucial component of transportation planning and management.

2.2 Researches based on Variational Autoencoder

In recent years, Variational Autoencoders (VAEs) have been widely used for approximate some complicated distributions.[15] The ability of VAEs has been proved to be promise in the works of generating many kinds of complicated data in image processing domain. However, some researches also using this framework in other domains such as Natural Language Processing (NLP), which inspired its implementation for tackling issues based on sequential data.

Y. Fan el at.[16] present a novel end-to-end partially sipervised deep learning approach for video anomaly detection and localization using normal samples. it is the first time that A Variational Autoencoder (VAE) framework utilized for video anomaly detection.

Y. Pu el at.[17] developed a novel Variational Autoencoder to model images, as well as associated labels or captions. They use a deep Convolutional Neural Network (CNN) as an image encoder, while A Deep Generative Deconvolutional Network (DGDN) is used as a decoder of the latent features. The proposed model achieve a high performance on image recognition.

Besides, there are many researches using Variational Autoencoders in Natural Language Processing (NLP). Jonas Muller el at.[18] implemented the Variational

Autoencoder framework for revising natural language sentences. Comparison between Variational Autoencoder and Encoder-Decoder models for short conversation is done by Shin Asakawa and Takashi Ogata.[19]

Another aspect of research based on Variational Autoencoder is improving the VAE framework itself. Sønderby, Casper Kaae, et al. proposed a Ladder Variational Autoencoders[20] which can recursively corrects the generative distribution by a data dependent approximate likelihood. Their moel can learn a deeper more hierarchy of latent variables than other generative model based on Variational Autoencoder.

A research about Infinit Variational Autoencoder is done recently.[21] They use a mixture model where the mixing coefficients are modeled by a Dirichlet process, allowing to integrate over the coefficients when performing inference. Their work shows the flexibility for the applications which have only a small number of available training samples.

2.3 Generative model for human mobility simulation

To simulate human behavior and moving patterns, various generative models have been developed in recent years.

Input-Output Hidden Markov Model (IO-HMM)[22] was proposed to enable activity based travel demand models which can protect the privacy of mobile phone users while using this cellular data to simulate synthetic agent travel patterns. Their model achieve a reasonable accuracy when conducting an agent-based microscopic traffic simulation. However, the limitation of the proposed model is that if travel patterns vary greatly over the region, a single model will not be able to capture all region with a good performance.

A Gibbs sampling based multiple hidden Markov model (GSMHMM)[10], designed as a part of city-coupling algorithm, can generate simulated trajectories in a city-wide scale area such as Tokyo or Osaka. However, as the model is based on Gibbs Sampling, it needs some important prior knowledge for the GSMHMM to generate new human mobility trajectories.

However, HMMs cannot completely model the temporal dependency of states. To improve the HMMs, Baratchi et al.[23] proposed Hidden Semi-Markov Model(HSMM), which including the duration of the state into the hidden variables. In general, their works are all based on Hidden Markov Model, and focus on reconstruct the trajectories of human mobility following a specific probability distribution.

Very recently, a non-Parametric generative model for human trajectories has been proposed.[24] They use Generative Adversarial Network (GAN) to produce data points after a simple and intuitive yet effective embedding for locations traces designed. It is the first time that deep learning methods implemented in building a generative model for human mobility in our knowledge.

The Sequntial Variational Autoencoder we build in this research has significant differences comparing with their model. Their work is a GAN based model which aims to generate fake data that can be recognized as true data by the trained discriminator. While the SVAE model in this research aims to learn the approximated latent distribution of training data first, then resample the fake data from this learned latent space. Besides, there is no need of trajectory transformation for trajectories when using SVAE model.

Chapter 3

Methodology

3.1 Autoencoder

3.1.1 Architecture of Autoencoder

Autoencoders (AE)[25] are unsupervised learning that aims to reconstruct their outputs in a way of making the outputs as similar as the inputs. They firstly compress the input into a latent space representation, which capture the features of the input, and then reconstructing the output from this informative representation. This kind of network is composed of two parts: 1) encoder: this is the part of the network that compresses the input into a latent space representation. It can be represented by an encoding function $h = f(x)$. 2) decoder: this part aims to reconstruct the input from the latent space representation. It can be represented by a decoding function $r = g(h)$.

The AE as a whole can thus be described by the function $r = g(f(x))$ where we want the reconstructed output r as close as the original input x.

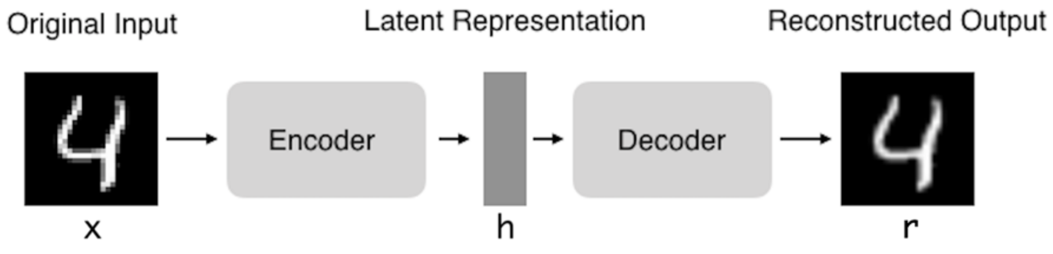


FIGURE 3.1: framework of autoencoders.

3.1.2 The purpose of Autoencoder

AEs will be useless if the only purpose of them was to copy the input to the output. Actually, what we want is that by training the AEs to reconstruct the outputs as similar as their inputs, the learned latent space representation h will take on useful properties. This purpose can be achieved by creating constraints on the copying task.

one way, which is called undercomplete, to obtain useful features of the inputs when training the AE is constrain h to have smaller dimension than the original inputs x. When training an undercomplete representation, what we actually do is forcing the AE to learn the most salient features of the original inputs and drop those redundant dimensionalities. However, the AE can learn to perform the copying task without extracting any useful information if it is given too much capacity.

It can occur if the dimension of the latent space representation is the same as the original inputs, and in overcomplete case, where the dimension of the latent space representation is larger than the original inputs. In these cases, it can be possible that even a linear encoder and a linear decoder can learn to reconstruct the outputs as similar as the original inputs without learning salient useful features about the training data distribution. In practical, we are able to train any architecture of AE successfully, choosing the code dimension and the capacity of the encoder and decoder based on the complexity of distribution to be approximated.

3.1.3 Applications of Autoencoder

Nowadays, it is told that two main interesting practical applications of AEs are Data Denoising[26] and Dimensionality Reduction[27] for data visualization. AEs are convinced to be more efficient in learning data projections than Principal Component Analysis (PCA)[28] or some other basic techniques if they are trained with appropriate dimensionality and sparsity constraints. AEs can be learned automatically from training data, which means that it is easy to train specialized instances of the algorithm that will give a good performance on a specific type of inputs and no new engineering is required, but only the appropriate training data.

However, AEs can hardly give a good performance in image compression. Since AEs are trained on a given set of data, a reasonable compression result will be achieved on the data which is similar to the original training data set used. But when the data set is not similar to the training data, it will perform bad anyway. It can be said that AEs cannot be a good robust general purpose image compressors. An advantage of AEs is that not only the AEs are trained to preserve as much information as possible, but also to make the new representation have various properties. Also, different kinds of AEs are designed to achieve different kinds of properties.

3.2 Variational Autoencoder

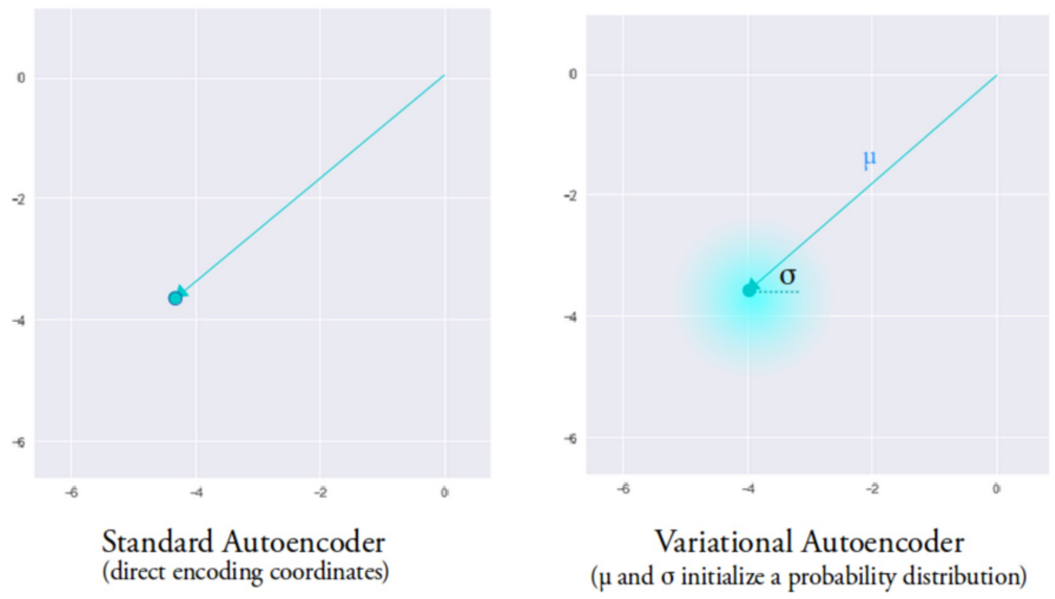
3.2.1 Variational Bayesian

Variational Bayes is a particular variational method which aims to find some approximate joint distribution $Q(x, \theta)$ over hidden variables x to approximate the true joint $P(x)$, and defines the distance as the Kullback-Leibler divergence $KL(Q(x, \theta)||P(x))$.[29] Kingma, Diederik P., and Max Welling[3] introduce a stochastic variational inference and leaning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case, and propose a Variational Autoencoder framework, which is widely used in recent years.

3.2.2 Architecture of Variational Autoencoder

Autoencoders is widely used for generation before, but its fundamental problem is that the latent space, constructed by the Autoencoder from learning the features of input data, may not be continuous, or allow easy interpolation. The purpose for building a generative model is that we want to randomly sample more data from the approximate latent space or generate variations on input data from a continuous latent space.

When the latent space constructed has discontinuities, the decoder will simple generate an unrealistic output if we sample or generate a variation from there. That is because the decoder lack the ability of dealing with that region of the



μ
σ
Standard Autoencoder
(direct encoding coordinates)
Variational Autoencoder
(μ and σ initialize a probability distribution)

FIGURE 3.2: comparison between Autoencoder and Variational Autoencoder.

latent space, since it never saw such kind of encoded vector from that region of latent space during training. One fundamentally unique property of Variational Autoencoders (VAEs) which separate them from vanilla Autoencoders is that their latent space is designed to be continuous, allowing easy random sampling and interpolation. It is also this property that makes Variational Autoencoders useful for generative modeling.

That property is achieved by making its encoder output two vectors of size n: a vector of means μ, and another vector of standard deviation σ, instead of just output one single encoding vector of size n. This two encoding vectors then form the parameters of a vector of random variables of length n, with the i-th element of μ and σ being the mean and standard deviation of the i-th random variable X_i, from which we sample to obtain the sampled encoding which we pass onward to the decoder. This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling. Figure 3.2 shows the different strategy of constructing latent space between standard Autoencoder and Variational Autoencoder.

Intuitively, there is a main difference between the constructed latent spaces of a standard Autoencoder and a Variational Autoencoder. In the latent space of a Variational Autoencoder, the encoded mean vector μ and the standard deviation σ initialize a probability distribution, while the encoded vector of a standard

Autoencoder is a direct encoding coordinate. In the case of training a Variational Autoencoder, encodings can be generated randomly from the probability distribution. Therefore, the decoder of a Variational Autoencoder can learn to reconstruct the output from a probability distribution rather than just a group of specific points in the latent space.

Kullback-Leibler divergence[30] is a measure of how one probability distribution diverge from a second, expected probability distribution. The most important metric in information theory is Entropy which is to quantify the information in data. The definition of Entropy for a probability distribution $p(x)$ is:

$$H = -\sum_{i=1}^{N} p(x_i) log p(x_i)$$

Based on the formula of entropy, the Kullback-Leibler divergence which measures the difference between a probability distribution $p(x)$ and the approximating distribution $q(x)$ can be given:

$$D_K L(p||q) = \sum_{i=1}^{N} p(x_i)(log p(x_i) - log q(x_i))$$

With Kullback-Leibler divergence we can calculate exactly how much information is lost when we approximate one probability distribution with another one.

The encoder of a Variational Autoencoder is designed to convert the input data point x to a hidden representation z, with weights and biases θ. Therefore, the encoder is denoted to be $q_\theta(z|x)$. The noisy values of hidden representation z is sampled from this distribution as the input of the decoder.The decoder of a Variational Autoencoder has weights and biases ϕ, denoted by $p_\phi(x|z)$. It get the noisy values of the latent representation z as input, and reconstruct the output data x.

The reconstruction log-likelihood $log p_\phi(x|z)$ is used to measure the information lost in aforementioned procedure. It also gives the efficiency of the decoder for reconstrcuting an input data x given its latent representation z.

The loss function of the Variational Autoencoder is:

$$l_i(\theta, \phi) = -E_{z \sim q_\theta(z|x_i)}[log p_\phi(x_i|z)] + KL(q_\theta(z|x_i)||p(z))$$

It contains two part: 1) the first term is named to be reconstruction loss; 2) the second term is a Kullback-Leibler divergence between the probability distribution of encoder and a unit Gaussian distribution. This loss function is well designed as we can also treat the second term to be a regularizer just like many other loss functions. A reconstruction loss is to force the model to give the output just as similar as possible comparing with input. Meanwhile, the purpose of the second term is to make sure the latent space constructed in the training process is not so complex. When the second term is small, we can achieve the goal of using a simple latent space to approximate the real posterior distribution of latent space.

3.2.3 The probability perspective of Variational Autoencoder

We begin with a dataset of observations x, which corresponding to a group of unobservable latent variables z. It is very important that infering the unobservable latent variables z using observations x to guess the real state of a situation. This is a process of calculating the posterior distribution $p(z|x)$ to infer the unobservable latent variables. The most basic formula for calculate $p(z|x)$ is:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Since observations is denoted by x, $p(x)$ is called evidence. Theoretically in mathematics, $p(x)$ can be calculated using $p(x) = intp(x|z)p(z)dz$. However, situations are very different in reality because aforementioned formula requires exponential time to be computed. Therefore, instead of calculating the evidence directly to obtain the real posterior distribution $p(z|x)$, finding a apporximate distribution of the real posterior distribution is a practical choice.

In variational inference, a Gaussian distribution is used for apporximating the real posterior distribution $p(z|x)$. We use $q_\lambda(z|x)$ denote the approximate distribution. As it is a Gaussian distribution, the latent variables can be given by the mean and variance $\lambda_{x_i} = (\mu_{x_i}), \sigma_{x_i}^2)$. Then the problem is that how can we measure the difference between the real posterior distribution $p(z|x)$ and the approximate distribution $q_\lambda(z|x)$. In information theory, the Kullback-Leibler divergence is often used for solving such problem:

$$KL(q_\lambda(z|x)||p(z|x)) = E_q[logq_\lambda(z|x)] - E_q[logp(x,z)] + logp(x)$$

By minimizing this divergence respect to the parameters λ, we can find the optimal approximate distribution. This process can be written as follows:

$$q^*_\lambda(z|x) =_\lambda KL(q_\lambda(z|x)||p(z|x))$$

However, aforementioned Kullback-Leibler divergence cannot calculated directly since the evidence $p(x)$ appears.

To tackle this problem, the solution comes to the Evidence Lower Bound (ELBO):

$$ELBO(\lambda) = E_q[logp(x,z)] - E_q[logq_\lambda(z|x)]$$

Combining the above Kullback-Leibler divergence and this ELBO function, we can get the formula for the evidence:

$$logp(x) = ELBO(\lambda) + KL(q_\lambda(z|x)||p(z|x))$$

According to the Jesen's inequality, the result of Kullback-Leibler divergence between two probability distribution always greater than or equal to zero, and zero can be achieved only when these two probability distribution are the same. Since the information of the evidence is a constant value, we can know that minimizing the Kullback-Leibler divergence is equivalent to maximizing the ELBO. As the result, the intractable problem is solved.

3.3 Sequence-to-sequence model

Deep neural networks that are mainly feedforward fully connected neural network are powerful but not really appropriate for sequential data such as time series data or language. They are very good to map input data to discrete output or continuous variable but not sequence to sequence mapping. Sequence-to-sequence (Seq2seq) model learns from variable sequence input fixed length sequence output. It uses two Long Short Term Memory (LSTM) model, one learns vector representation from input sequence of fixed dimensionality an another LSTM learns to

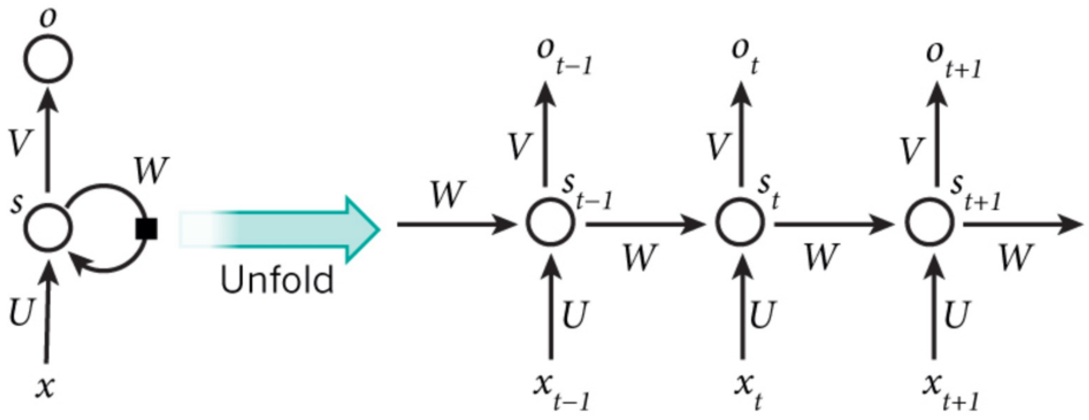


FIGURE 3.3: architecture of Recurrent Neural Network.

decode from this input vector to target sequence. LSTM is a variant of recurrent neural network that solves problem of handling long sequences using different gates. Seq2seq model was recently proposed, and demonstrated excellent result for Natural Language Processing (NLP).[31–33] This model proved to be more effective than previous methods at NMT, and is apparently now used by Google Translate.

3.3.1 Recurrent Neural Network

Recurrent Neural Networks (RNN) are widely used in solving many sequential problems such as Natural Language Processing (NLP) tasks.[34–36] The main contribution of RNNs is that they can capture the sequntial information for use. For instance, it is good idea to obtain the previous location which a data point located in before we make a prediction of the next location where the data point will be. A typical RNN is shown as figure 3.3.

Input data is denoted by $x = (x_1, ..., x_{t-1}, x_t, x_{t+1}, ...)$. An observation x_t indicates the observed data in step t. Corresponding to the input data, a hidden space is denoted by $s = (s_1, ..., s_{t-1}, s_t, s_{t+1}, ...)$. However, a hidden space is not only related with input data but also related with previous hidden state: $s_t = f(Ux_t + Ws_{t-1})$. The functin $f(\cdot)$ is nonlinear activation such as ReLU or tanh. A hidden state s_t can capture the information of current observation x_t and take the previous information captured by s_{t-1} into account. Then, the output $o = (o_1, ..., o_{t-1}, o_t, o_{t+1}, ...)$ can be calculated by $o_t = g(Vs_t)$, where function $g(\cdot)$ is another nonlinear activation.

From figure 3.3, we notice that weights U, V, and W only be changed after a sequence be computed completely. Therefore, the total numbder of parameters of a Recurrent Neural Network is not so big comparing with other traditional deep neural networks. However, there is a main difference between Recurrent Neural Network and some other traditional deep neural networks, which is that the backpropagation algorithm is used for training a traditional neural networks while it cannot be used for training a Recurrent Neural Network. That is because the gradient at each output depends on not only in current step, but also previous steps. In that case, a specific backpropagation is designed for traing a Recurrent Neural Network which is called Backpropagation Through Time (BPTT).

3.3.2 Backpropagation Through Time

Backpropagation Through Time (BPTT) is designed by Werbos, Paul J.[37] It can be used in not only for training Recurrent Neural Networks, but also in other networks. We will give a brief introduction about the Backpropagation Through Time used in training Recurrent Neural Network. The most basic equations of RNNs is as follows:

$$s_t = f(Ux_t + Ws_{t-1})$$

$$o_t = g(Vs_t)$$

The cross-entropy loss is often used for training a Recurrent Neural Network, the formula is given by:

$$E_t(o_t, \hat{o}_t) = -o_t log \hat{o}_t$$

$$E(o, \hat{o}) = \sum_t E_t(o_t, \hat{o}_t) = -\sum_t o_t log \hat{o}_t$$

In above equations, o_t is the ground truth at step t, while $\hat{o}_t$ is the prediction at step t. We use the sum of cross-entropy loss of all steps to measure the loss of the entire sequence.

The optimal results can be obtained when the cross-entropy loss is minimized, which is the common goal of training most of neural networks. In previous part, we have already explain the process of how a Recurrent Neural Network can output predictions o_t give input x_t. If we want to reduce the error between inputs and outputs, the key problem is to find how much contribution of parameters U, W, and V respectively for the loss. Using Stochastic Gradient Descent would be the solution for calcaute the gradient of the loss respect to different parameters. Just like sum up the loss in the defined cross-entropy loss functin, we can sum up the gradient at each step:

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

These gradient can be calculated using a chain rule of differentiation. We set $z_t = Vs_t$, then we can calculate the gradient:

$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{o}_t}\frac{\partial \hat{o}_t}{\partial V} = \frac{\partial E_t}{\partial \hat{o}_t}\frac{\hat{o}_t}{\partial z_t}\frac{\partial z_t}{\partial V}$$

Note that $s_t = f(Ux_t + Ws_{t-1})$ depends on s_{t-1}, so when we take derivative with respect to W and U, the formula of gradients respect to W and U is a bit different:

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{o}_t}\frac{\partial \hat{o}_t}{\partial s_t}\frac{\partial s_t}{\partial s_k}\frac{\partial s_k}{\partial W}$$

$$\frac{\partial E_t}{\partial U} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{o}_t}\frac{\partial \hat{o}_t}{\partial s_t}\frac{\partial s_t}{\partial s_k}\frac{\partial s_k}{\partial U}$$

We can solve the equation use aforementioned equations.

3.3.3 The Vanishing Gradient Problem

RNNs have difficulties learning long-range dependencies. We should explain this difficulty using aforementioned gradient calculation. The problem is caused by $\frac{\partial s_t}{\partial s_k}$ in above equations. The formula $\frac{\partial s_t}{\partial s_k}$ is a cahin rule in itself, which is the derivative of a vector function with respect to a vector. the result of this formula can be written as a Jacobian matrix, then the gradient can be written as follows:

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{o}_t}\frac{\partial \hat{o}_t}{\partial s_t}\left(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}}\right)\frac{\partial s_k}{\partial W}$$

As the activation function we often use in a Recurrent Neural Network cell is tanh:

$$f(x) = \frac{2}{(1+^{-2x})} - 1$$

$$f'(x) = 1 - f(x)^2$$

the tanh function maps all input x into a value ranged between -1 and 1, its derivative is also bounded by 1. Besides, the derivative of the tanh function is 0 at both ends, which means that a zero gradient will obtained in some specific cases. Once a zero gradient is obtained by chance, it will drive other gradient towards zero since the chain rule of the gradient. Therefore, the gradient vanishing problem occurs when the values of these gradient shrinking exponential fast even start with some small values.

There are some solutions for preventing the gradient vanishing problem. A ReLU activation is introduced[38], of which derivate is a constant:

$$f(x) = max(0, x)$$

Moreover, Long-Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are also invented owing to this gradient vanishing problem.

3.3.4 Long-Short Term Memory

Long-Short-Term Memory (LSTM) was designed to combat vanishing gradients through a gating mechanism.[39] How a LSTM calculates a hidden state s_t is shown as follows:

$$i = \sigma(x_t U^i + s_{t-1} W^i)$$

$$f = \sigma(x_t U^f + s_{t-1} W^f)$$

$$o = \sigma(x_t U^o + s_{t-1} W^o)$$

$$g = \tanh(x_t U^g + s_{t-1} W^g)$$

$$c_t = c_{t-1} \circ f + g \circ f$$

$$s_t = \tanh(c_t) \circ o$$

A LSTM layer, shown in figure 3.4, has three gates i, f, o. i is called input gate, f is output gate, and o is output gate. The sigmoid function is used in thse gates which has values between 0 and 1. For example, if the value of a gate is 1, then it means that let all information pass towards, while if the value of a gate is 0, it

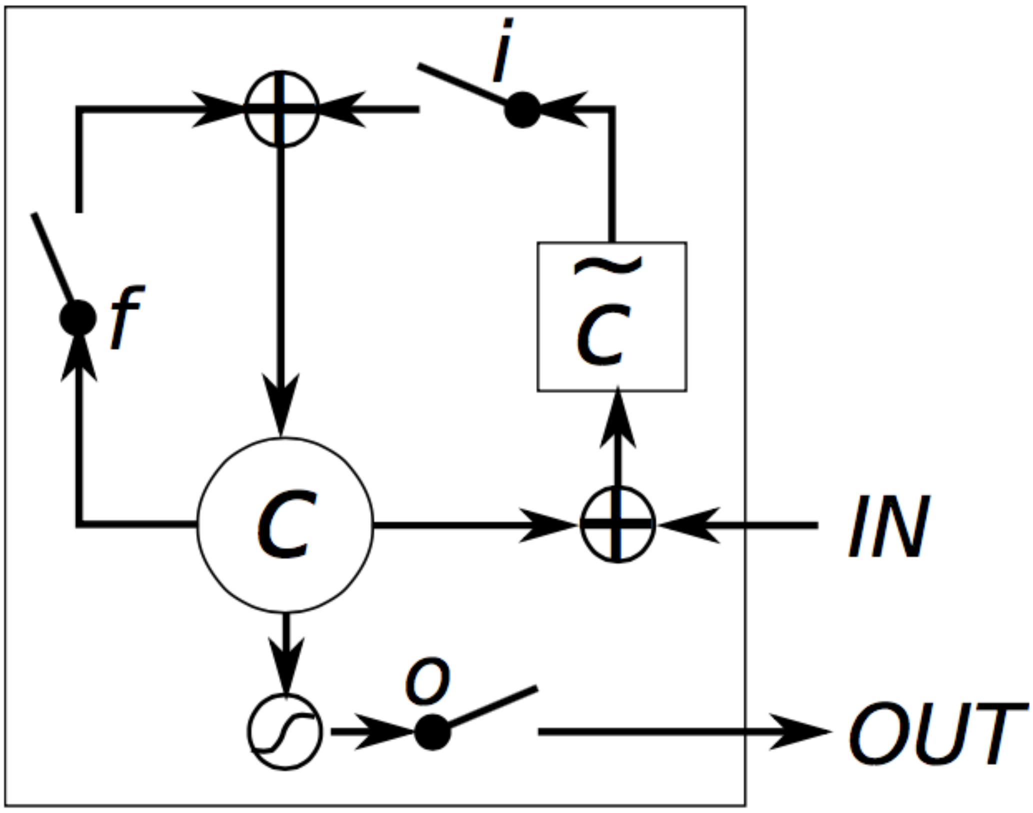


FIGURE 3.4: architecture of Long-Short-Term Memory (LSTM).

means that no information shall passed onwards. The function of different gates is different. The input gate i determines the quantity of information of current input to be passed onwards. The forget gate f determines the quantity of information of previous state to be passed onwards. The output gate o determines the quality of information of internal state to be passed onwards.

Besides, g is designed to be a kind of candidate hidden state which is also calculated based on the current input x_t and the previous output s_{t-1} just like the hidden state calculated in a vanill Recurrent Neural Network. However, This candidate hidden state is not the final hidden state calculated in a Long-Short-Term Memory as it should be selected by the aforementioned input gate.

c_t in a Long-short-Term Memory unit is the internal memory. It is used to capture the information combining the previous internal memory c_{t-1} with selected candidate hidden state g. Using a internal memory, we can completely ignore the previous memory by set the value of the forget gate to be 0, or completely ignore the new input by set the value of input gate to be 0. However, what we really want is information between these two extremes.

Finally, we can compute the output hidden state s_t using the internal memory c_t. Since there is a output gate which control the quantity of information to be passed onwards, the hidden state s_t could contain only part information of the internal hidden state. The ability of modeling long-term dependencies is improved in LSTMs thanks to the gating mechanism.

3.3.5 Gated Recurrent Unit

Gated Recurrent Unit (GRU) is another gating mechanism in RNNs, introduced by Kyunghyun Cho el at.[40] Equations for calculating the hidden state in a Gated Recurrent Unit is given:

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^z + s_{t-1} W^r)$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

A GRU, shown as figure 3.5, has only two gates r and z. r is called reset gat, while z is called update gate. Intuitively, the reset gate is used to guide the combination of the input and the previous memory, and the update gate is used to guide how much information of previous memory should keep.

Although Gated Recurrent Unit and Long-Short-Term Memory share the same idea of using a gating mechanism, there are some important differences: 1) GRUs only have two gates while LSTMs have three; 2) there is no second nonlinear activation applied in GRUs when computing the output.

According to evaluations conducted in some recent research,[41, 42] there is no clear whiner when conducting a comparison between GRUs and LSTMs. It seems to be that if we have enough data, LSTMs could have better results. Meanwhile, GRUs can be trained faster since they have fewer parameters.

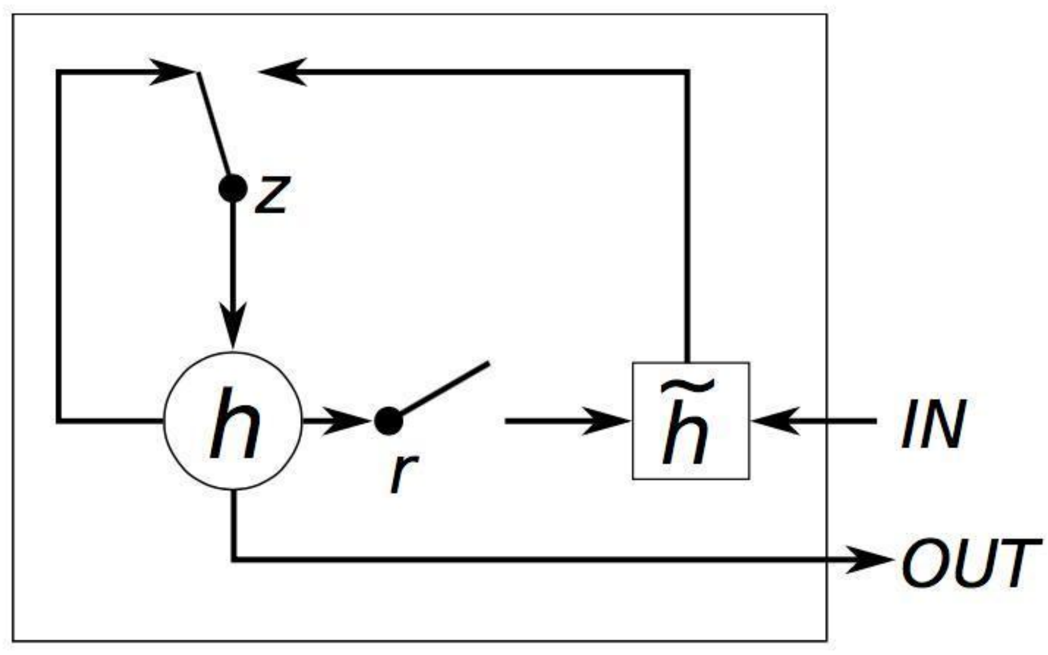


FIGURE 3.5: architecture of Gated Recurrent Unit (GRU).

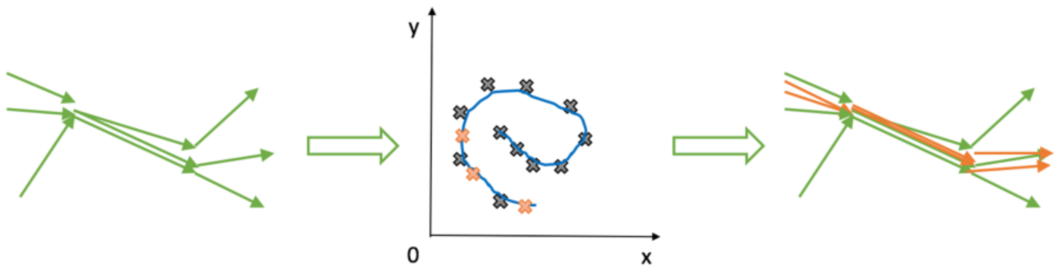


FIGURE 3.6: The idea of implementing generative model for trajectories of human mobility. Left penal of the figure is some examples of observed trajectories of human mobility; middle one is a example of approximate distribution of t rajectories of human mobility; right one is reconstrcuted trajectories.

3.4 Sequential Variational Autoencoder

The idea of using generative model to handle the incomplete data is very natural as figure 3.6 shows. When we have some trajectories of human mobility obtained from incomplete data set, we can firstly use an encoder to encode the trajectories into a low dimensional points in hidden space, shown as blue points in above figure. Then we can find an approximate distribution of these points, shown as blue line in the above figure. it is natural to sample some points from the learned distribution, and use a decoder to reconstruct the points in the hidden space to

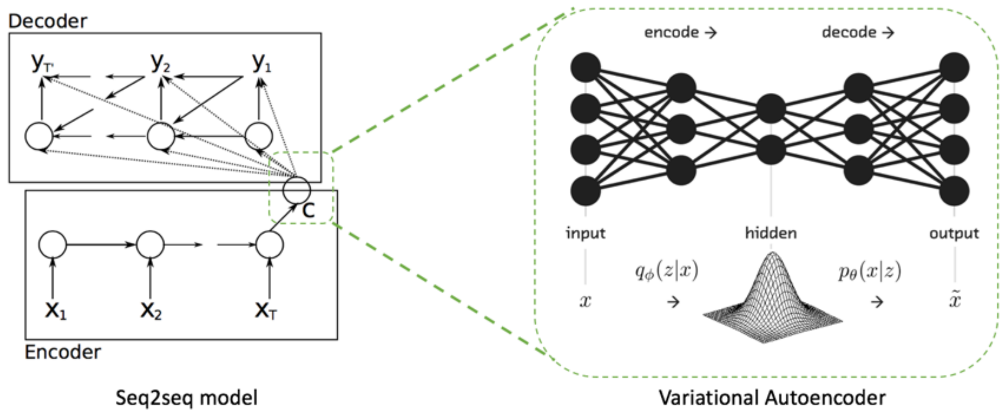


FIGURE 3.7: A graphical model visualization of proposed Sequential Variational Autoencoder. Left penal shows a seq2seq model framework, right penal shows a variational autoencoder framework.

the trajectories we want. From an intuitive view, it is easier to generate new data from a lower dimensional distribution. Also, the generated new fake trajectories are reasonable as they are resampled from the learned distribution based on the real trajectories.

As discussed in the above, A Variational Autoencoder can build a hidden space which follow Gaussian distribution to approximate the real distribution of the observed trajectories. The reason for a constructing a hidden space which follows a Gaussian distribution is that by learning the parameters of the Gaussian distribution representing the input observed trajectories, we can sample from the distribution and generate new samples of trajectory. The ability for constructing hidden space following a Gaussian distribution is exactly what we want in our proposed Sequential Variational Autoencoder. However, the Variational Autoencoder lack the ability of tackling sequential data, which is the main limitation.

A seq2seq model framework usually use several Recurrent Neural Networks as encoder and decoder. Therefore, a seq2seq model can handle sequential data without difficulties, but the hidden space C is not well constructed.

We can regard the seq2seq model as a Sequential Autoencoder. By doing that, it is natural to consider that if we combine Variational Autoencoder and seq2seq model as figure 3.7 shows, we can combine their advantages. That means the Sequential Variational Autoencoder is well-designed generative model for sequential data.

Let $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_t})$ denote a high dimensional sequence, such as a trajectory of human mobility wit t steps. We use a LSTM neural network as recurrent encoder to capture the information of the input trajectory $\mathbf{x}$. Then we will obtain a series of hidden state s_t, and a series of output o_t. In actual case, what we really care about is the final output o rather than a sequence of output value o_t. Since we only keep the final output, we can obtain a intermediate non-sequential vector o to represent the information captured from the input sequence using this recurrent encoder. After intermediate vector o is obtained, we treat this vector as the input of the Variational Autoencoder part. Then we can write the joint probability of the model as $p(o, z) = p(o|z)p(z)$. $p(z)$ is a prior latent distribution, and $p(o|z)$ is the likelihood. Then we need to calculate the posterior latent distribution $p(z|o)$ given observed data:

$$p(z|o) = \frac{p(o|z)p(z)}{p(o)}$$

by marginalizing out the latent distribution:

$$p(z|o) = \frac{p(o|z)p(z)}{\int p(o|z)p(z)dz}$$

This is a exponential time-consuming process. Therefore, variational inference approximates the real posterior distribution with a family of distribution $q_\lambda(z|o)$. Usually, we choose q to follow a Gaussian distribution, then λ would be the mean and variance of the latent distribution $\lambda = (\mu, \sigma)$. Kullback-Leibler divergence is used for measuring the information lost when using q to approximate p, the optimal approximate posterior is thus:

$$q^*_\lambda(z|o) = argmin_\lambda KL(q_\lambda(z|o)||p(z|o))$$

In the SVAE model, we parametrize approximate posterior $q_\theta(z|o)$ using an inference network, approximate likelihood $p_\phi(o|z)$ using a generative network. Then the loss of the model will be:

$$loss = -E_{q_\theta(z|o)}[logp_\phi(o|z)] + KL(q_\theta(z|o)||p(z))$$

Finally, we use another LSTM neural network as recurrent decoder to reconstruct

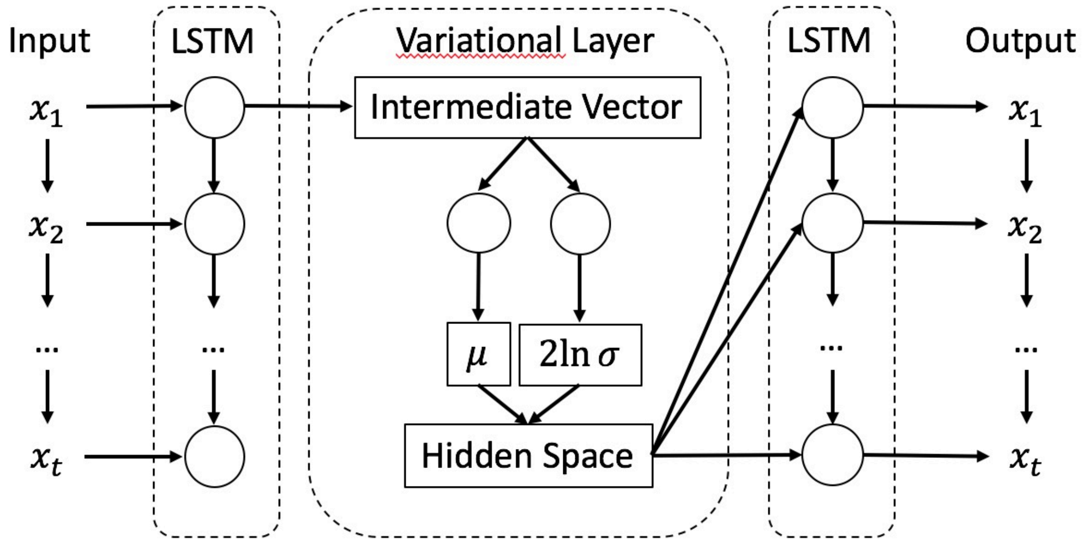


FIGURE 3.8: workflow of the training procedure of the SVAE.

the trajectories of human mobility, $\mathbf{x}$ from parameters in learned latent distribution. Training of the model is shown as figure 3.8

Chapter 4

Experiment

4.1 Preliminary

4.1.1 Description of raw data

The data we used for this research is navigation locational data, which is collected when vehicles were using navigation application. The coordinate system of this GPS data is WGS84, and the records of the locations cover all over Japan. However, owing to some reasons, such as privacy protection, we can only use one month records which is from Oct 1, 2015 to Oct 31, 2015. Besides, the ID of the users were deleted, so the privacy is protected well. We can only get the information of the ID of each navigation route to distinguish different trajectories. Our data contains the information of:
1) Daily user ID: a random unique ID of a vehicle in a day;
2) Route ID: the unique ID of each navigation trip;
3) Timestep: the recorded time of current location;
4) Longitude and Latitude: the value of longitude and latitude after conducted map matching;
5) Sensor longitude and Sensor latitude: the value of raw records of longitude and latitude.
To get an intuitive image of the data we used, a visualization of the GPS data in selected Tokyo area is given as follows:

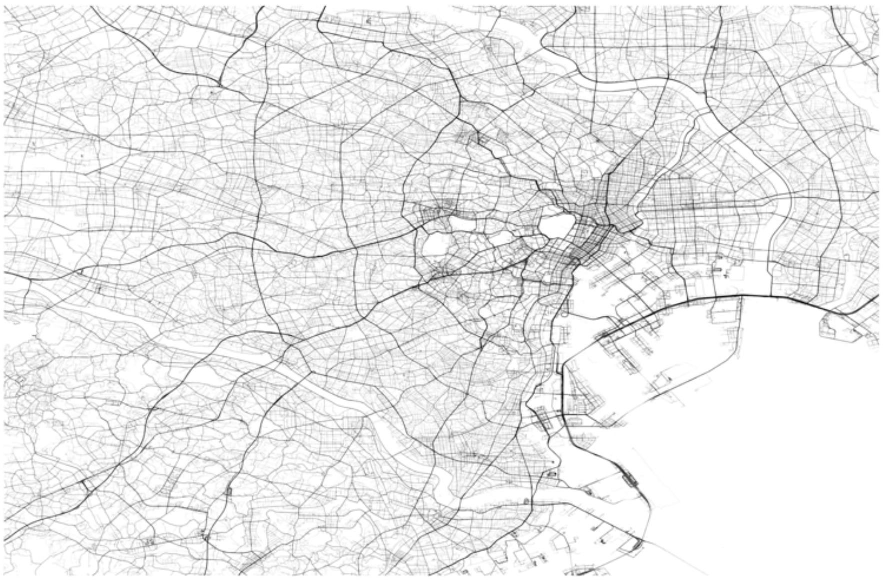


Figure 4.1: Distribution of navigation GPS points of NAVITIME

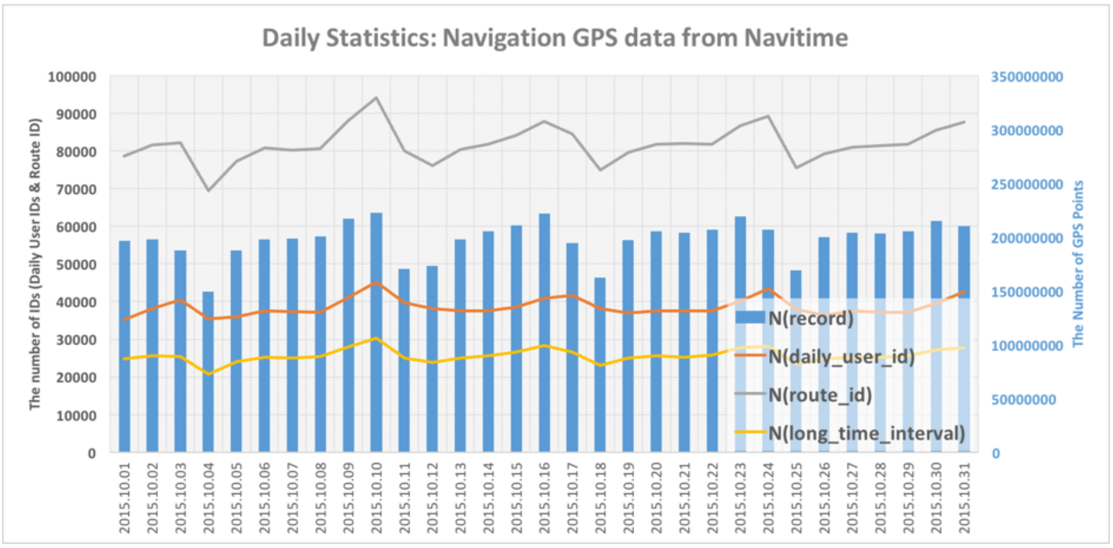


Figure 4.2: Daily statistics.

The figure 4.1 given is drawn using the recorded locational points. Since the records is dense and map matched, the points can shape lines and infer the road map perfectly. Moreover, we can imagine intuitively, more vehicles drive in major road than those drive in a small road, thus we can see that the lines of major roads are thicker than small roads. We conduct daily statistics of the used navigation GPS data:

In summary of the figure 4.2 shows, we get a simple table 4.1:

TABLE 4.1: summary of daily statistics

Total records	6,137,308,784
Total daily user IDs	1,168,592
Total route IDs	2,507,308
Average records	197,977,703/day
Average daily user IDs	38,632/day
Average route IDs	81,791/day

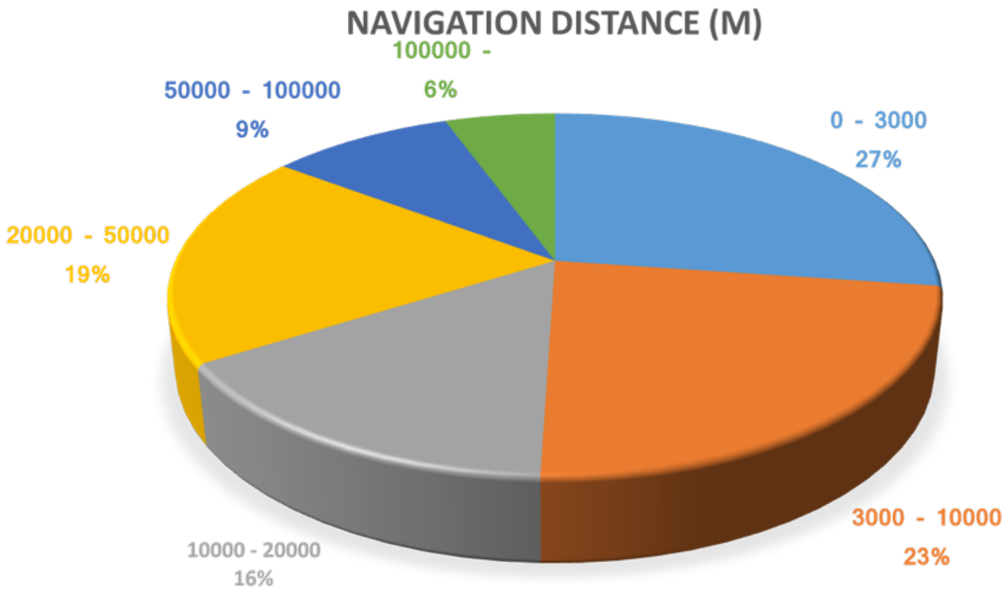


FIGURE 4.3: Chart of ratio of navigation distance

There is a pattern we can get from the histogram above which is that records in weekends is less than records in weekdays. We also interested in the navigation distance of the navigation GPS data shown in figure 4.3.

We can see that about the distance of 85% of navigation trips is shorter than 50,000 meters, so we can conclude that most navigation can be regarded as short trip navigation. Therefore, this navigation GPS data is suitable for some researches tackle city-scale problem. Another point of the navigation GPS data is the accuracy, shown in figure 4.4:

The blue rectangles present the sensor longitude and latitude which is the original position data and the yellow triangles present the longitude and latitude which is the position data after map matching. Both raw records and map matched records are very accurate. Besides, from right panel of the figure, some locational point is drawn. We can see a perfect trajectory with clear origin and destination,

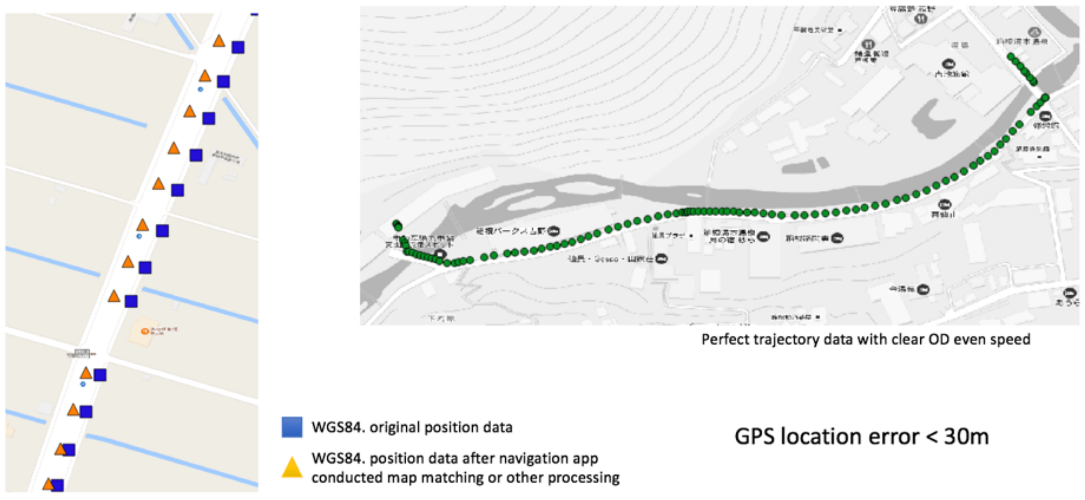

Perfect trajectory data with clear OD even speed
WGS84. original position data
WGS84. position data after navigation app conducted map matching or other processing
GPS location error < 30m

FIGURE 4.4: accuracy of the navigation GPS data.

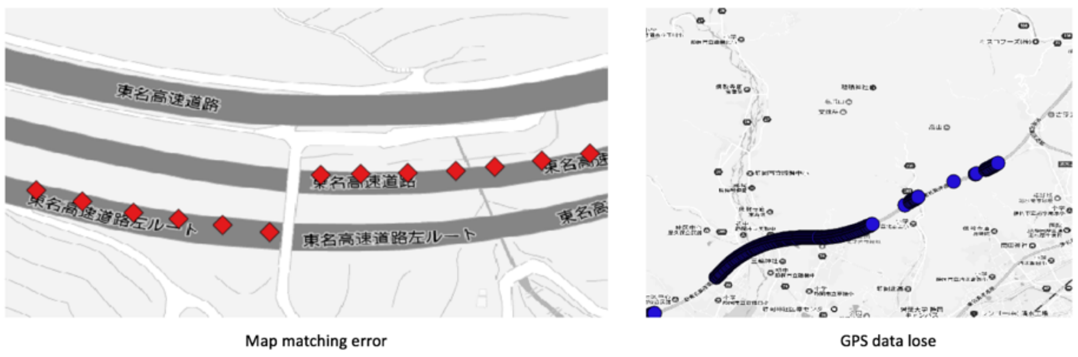


FIGURE 4.5: limitations of the navigation GPS data.

moreover, we can even get an intuitive driving speed by the distribution of the points. However, the data is not always so accurate.

There is also some map matching error or some GPS data records lose in a navigation trip shown as figure 4.5. But those small errors will have small influence on experiment results since we will conduct data preprocessing rather than just use the raw data.

4.1.2 Data preprocessing

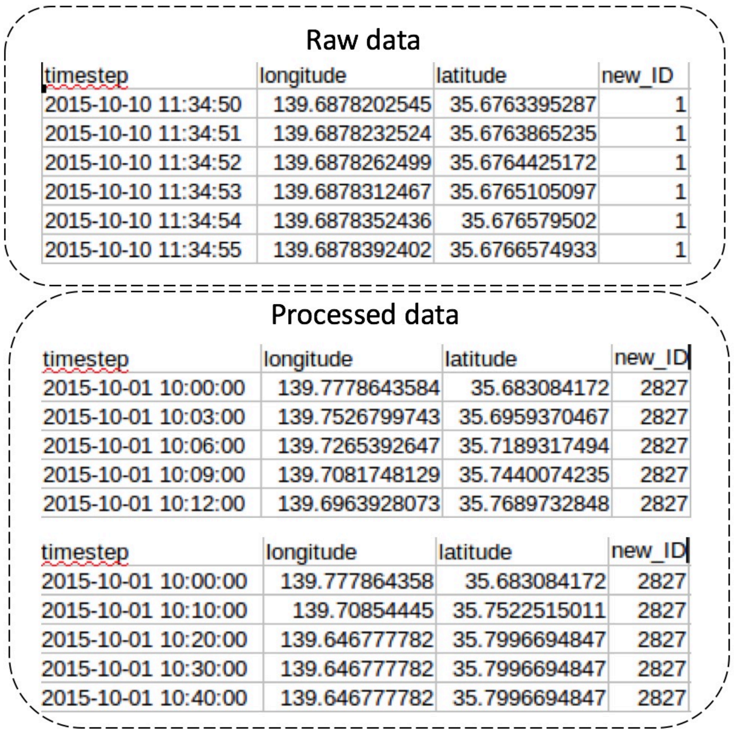
The navigation GPS data we used in this research is a really big data and contains a wealth of sequential information. However, it is very difficult to handle such big data, we must do some data preprocessing for this raw data then get a data set we want to utilize in our experiment. The aforementioned basic statistics of the navigation GPS data is all done by coding using python. Since the whole data is as huge as 1.2 terabyte, divided into 938 common-separated values (csv) files, conducting statistics on such big data is very hard time-consuming work. To improve the efficiency of basic statistics, we use parallel computing to make full use of central processing units of my machine. Thus, the computing time is reduced to one sixth and save lots of time. We use the "haversine" formula to calculate the great-circle distance between two points, which is the shortest distance over the earth's surface.

$$a = \sin^2(\delta\phi/2) + \cos\phi_1 \cos\phi_2 \sin^2(\delta\gamma/2)$$
$$c = 2a\tan(\sqrt{a}\sqrt{(1-a)})$$
$$d = Rc$$

Where ϕ is latitude, γ is longitude, R is earth's radius (mean radius is 6,371 km). Then, we get distance delta between each two points using above algorithm. By summarizing the distance delta of the same navigation trip, we can finally get the traveling distance of all trajectories in the navigation GPS data.

Another processing is that we also compute the time interval between each two points, although it is not used in aforementioned basic statistics, but it will be useful for the experiment. As the time interval of the raw data is not fixed, which means that it will lead to some potential difficulties to further use.

To simplify the data structure of the data which we will use in the experiment, we conduct a linear interpolation to the navigation GPS data to make the time interval of the records fixed. The reason for a linear interpolation is two-fold: 1) simplify the data structure; 2) obtain trajectories in a specific length. The navigation GPS data is not intuitive for those who are not familiar with trajectory data, so the visualization of the navigation GPS data is necessary. The visualization tool is called mobmap developed by Satoshi Ueyama, a researcher from our laboratory. In this paper, we use mobmap to visualize both the raw GPS data and the output results of our proposed model to make the data and result more intuitive.



Raw data

timestep	longitude	latitude	new_ID
2015-10-10 11:34:50	139.6878202545	35.6763395287	1
2015-10-10 11:34:51	139.6878232524	35.6763865235	1
2015-10-10 11:34:52	139.6878262499	35.6764425172	1
2015-10-10 11:34:53	139.6878312467	35.6765105097	1
2015-10-10 11:34:54	139.6878352436	35.676579502	1
2015-10-10 11:34:55	139.6878392402	35.6766574933	1

Processed data

timestep	longitude	latitude	new_ID
2015-10-01 10:00:00	139.7778643584	35.683084172	2827
2015-10-01 10:03:00	139.7526799743	35.6959370467	2827
2015-10-01 10:06:00	139.7265392647	35.7189317494	2827
2015-10-01 10:09:00	139.7081748129	35.7440074235	2827
2015-10-01 10:12:00	139.6963928073	35.7689732848	2827

timestep	longitude	latitude	new_ID
2015-10-01 10:00:00	139.777864358	35.683084172	2827
2015-10-01 10:10:00	139.70854445	35.7522515011	2827
2015-10-01 10:20:00	139.646777782	35.7996694847	2827
2015-10-01 10:30:00	139.646777782	35.7996694847	2827
2015-10-01 10:40:00	139.646777782	35.7996694847	2827

FIGURE 4.6: comparison between raw data and processed data.

4.1.3 Description of training data

For creating the data set used in our experiment, not all raw data is necessary as the size of the raw navigation GPS data is too big. Instead, we choose a selected Tokyo area, longitude from 135.5 to 139.9 and latitude from 35.5 to 35.8. Also, it is not necessary to use the whole month GPS data since the most navigation distance is shorter than 50,000 meters and will be ended in one single day. Since most of navigation trip will last hours, it is natural to get one hour's data to conduct the experiment.

We select the records of from 10 am to 11 am October 1, 2015. The data set contains more than 2,000 trajectories, which has fixed time interval. The visualization of the data set is shown as figure 4.6.

4.2 Experimental settings

The experiment is conducted as figure 4.7 shows. We make a brief description about the general process of how to train the SVAE model. The first step is preparing training data. The input data we used in the experiment is navigation GPS data which contains trip ID, longitude, latitude, and timestamp. However, the raw data should be preprocessed before the training process. The data preprocessing of linear interpolation is done to simplify the input data, by forcing the trajectories have fixed timestamp. Therefore, the input only contains information about longitude and latitude but can still represent the dynamics of the trajectories.

We then use several LSTMs as a recurrent encoder which aims to capture the salient features of the input sequential data. LSTMs return an output in every step, which means that the output could also be a sequential output. However, in the SVAE model, a non-sequential output, which we make it a intermediate vector, is better. This intermediate vector captures the salient features of the input trajectories while keeps a non-sequential data structure. We want the intermediate vector to be non-sequential since the custom variational autoencoder has no ability to handle the sequential data.

After the intermediate vector is given by the recurrent encoder, it will be the input of the custom variational autoencoder. This layer aims to build the latent space which can capture the features of the input and follow a Gaussian distribution at the same time. The output of this layer is mean and logarithm variance which are used for constructing the latent space which follows the Gaussian distribution. The final output of this layer is sampled from this latent space, and it will be the input of next recurrent decoder.

The latent vector should be repeated several times to match the length of the output trajectories. Then the recurrent decoder consisted of several LSTMs will reconstruct the output trajectories using aforementioned latent vector. Reconstructed trajectories should be as similar as possible comparing with input original trajectories by minimizing the loss function. At the same time, the latent Gaussian distribution should also be as simple as possible to make the SVAE model robust.

We use aforementioned data to conduct experiment. Mean Distance Error (MDE) between real trajectories and generated trajectories is used for evaluating the performance of the SVAE model with different parameter settings:
(1) Short sequence and long sequence, of which length is 6 and 20 respectively, as input of the SVAE model to test the ability for tackling long sequence of the

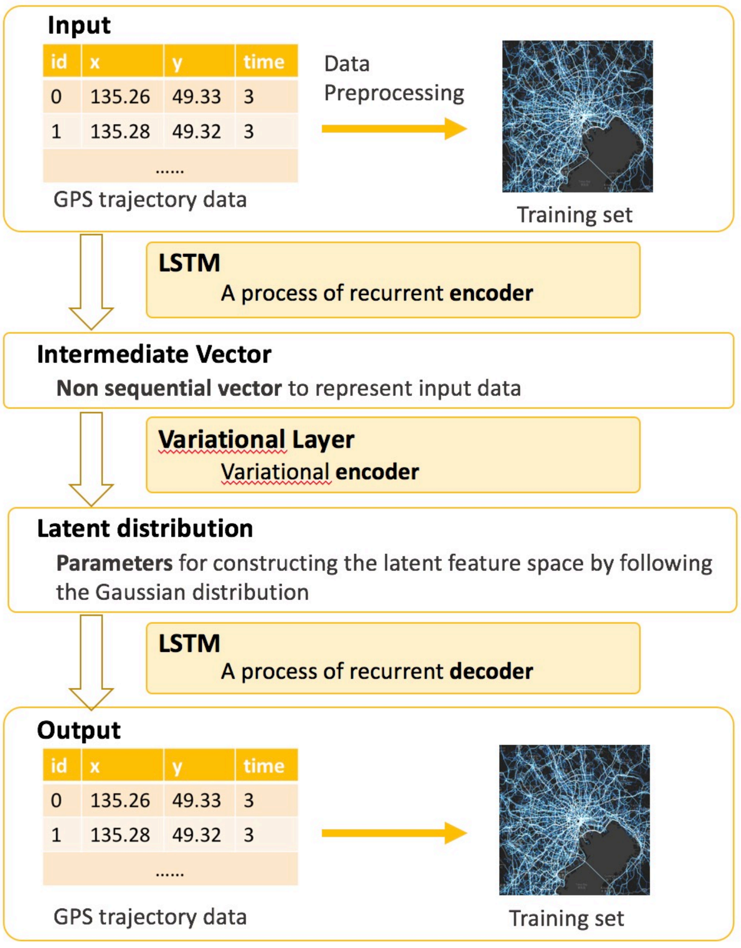


FIGURE 4.7: Training procedure of SVAE.

model;
(2) The dimensionality of hidden space is set to be 8, 12, 16 respectively to test the performance of the model for different dimensionality of hidden space;
(3) Three kinds of input (values of coordinate only, grid ID only and combination input of values of coordinate and grid ID) are tested.
The results are given in next section.

4.3 Experimental results

4.3.1 Results and Visualization

We use two datasets as our training set of SVAE model. One dataset is 2,000 trajectories of which length is all set to be 6, and the other one is 2,000 trajectories of which length is all set to be 20. The two data set is all chosen from the same

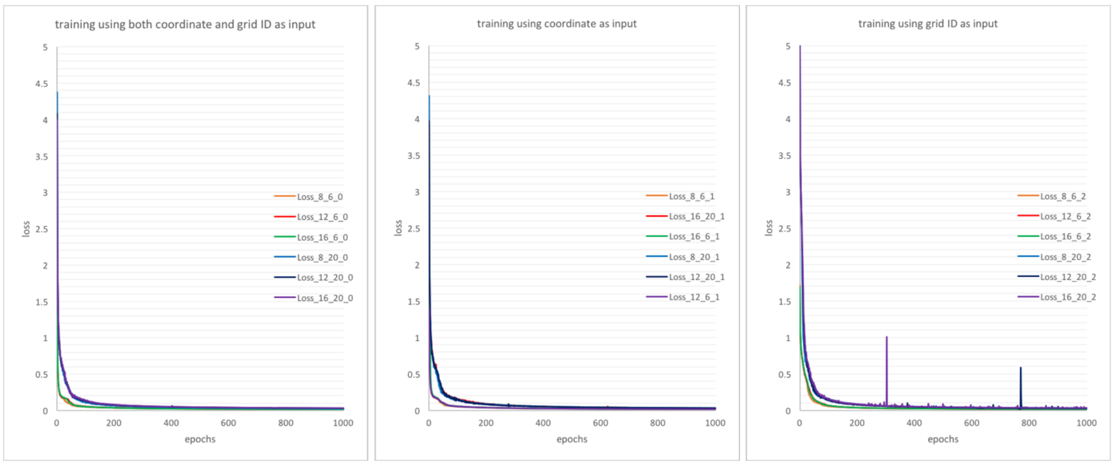


FIGURE 4.8: Training process.

TABLE 4.2: SVAE Loss

Loss	both input	coordinate input	grid ID input
6 steps, 8 latent dimension	0.017318176	0.018159691	0.017484304
20 steps, 8 latent dimension	0.027957876	0.02932067	0.027624224
6 steps, 12 latent dimension	0.017548803	0.018433879	0.01732461
20 steps, 12 latent dimension	0.027994325	0.030799899	0.029310457
6 steps, 16 latent dimension	0.017232143	0.018182858	0.017890416
20 steps, 16 latent dimension	0.029631174	0.031207314	0.03502047

raw dataset, but with different length of every sequence. Respectively, we train the SVAE model using these two datasets, changing the parameters which controls the dimensionality of the constructed latent space, and inputs.

Training processes of different parameter settings of model have been recorded as figure 4.8. The left part of the figure is the training process using both coordinate values and grid ID as input of the SVAE model. The middle part is records of training just using coordinate values as input. The right part is records of training just using grid ID as input. We set the training epochs of all different models as 1,000, which aims to make sure the all different raining process were finished.

The values of loss in different SVAE models have been summarized in the table 4.2. The values of the loss is calculated using aforementioned formula:

$$loss = -E_{q_\theta(z|o)}[logp_\phi(o|z)] + KL(q_\theta(z|o)||p(z))$$

The values in the table is given by the loss of final step's training. The smaller the value is, the better the results of the trained SVAE should give theoretically.

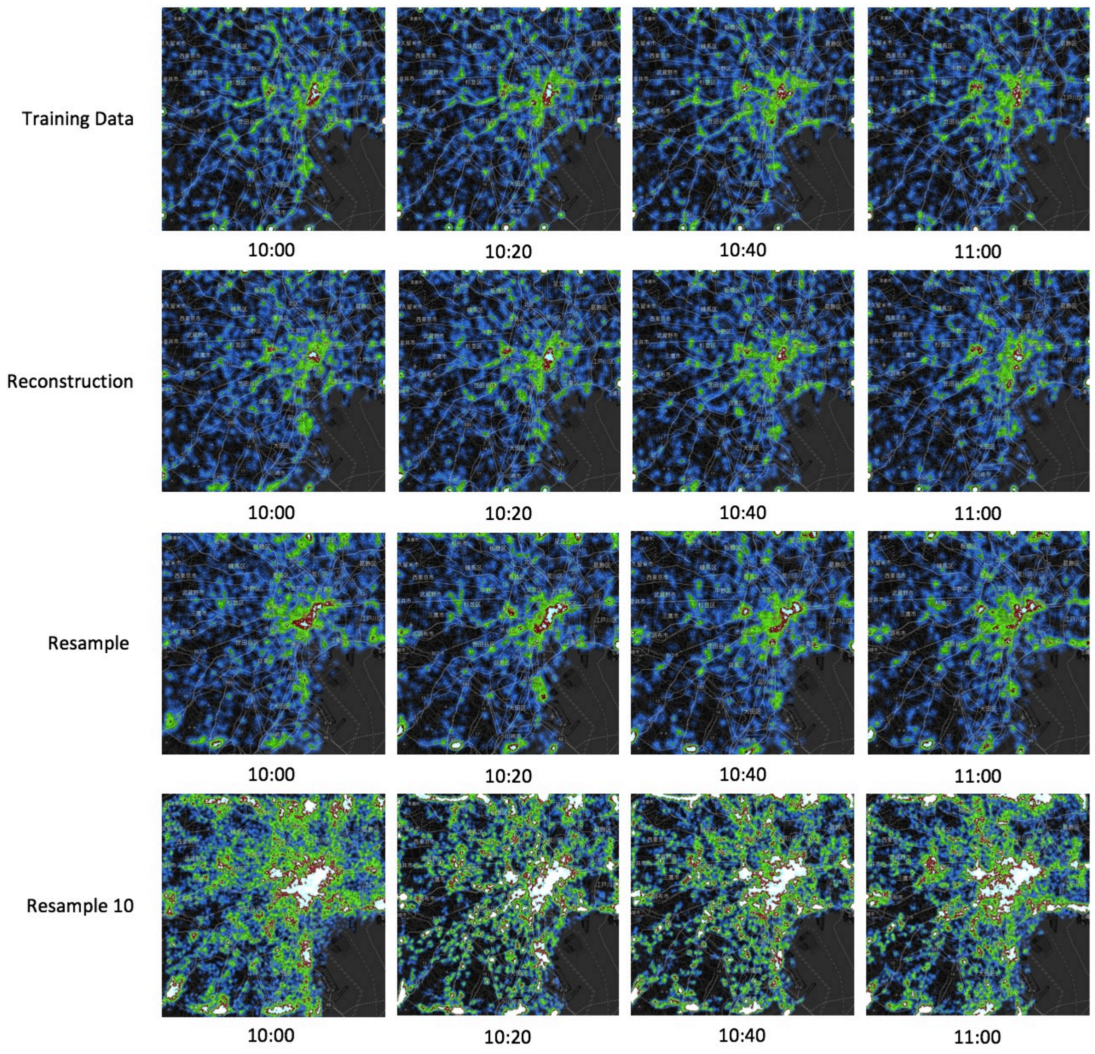


FIGURE 4.9: Visualization of training data, reconstrcuted trajectories, resampled trajectories and 10 times of resampled trajectories.

To make the results be understood easier, the figure 4.9 is given. This figure contains four rows and four columns of pictures because that we need show the visualization of dynamics of trajectory of human mobility. The first row of figure is four pictures about the training data, which is also the ground truth; the second row is reconstructed results, which reflects the performance of copying the original input training data to output data; the third row is resampled trajectories of human mobility, of which number is the same as the training data but generated from learned hidden space directly; the fourth row is the ten times resampled trajectories, which has more trajectories than original input training data.

Intuitively, we have confidence to say that the SVAE model has a good performance of modeling the input trajectories and reconstructing them. The patterns of resampled trajectories is a little different comparing with training data but still

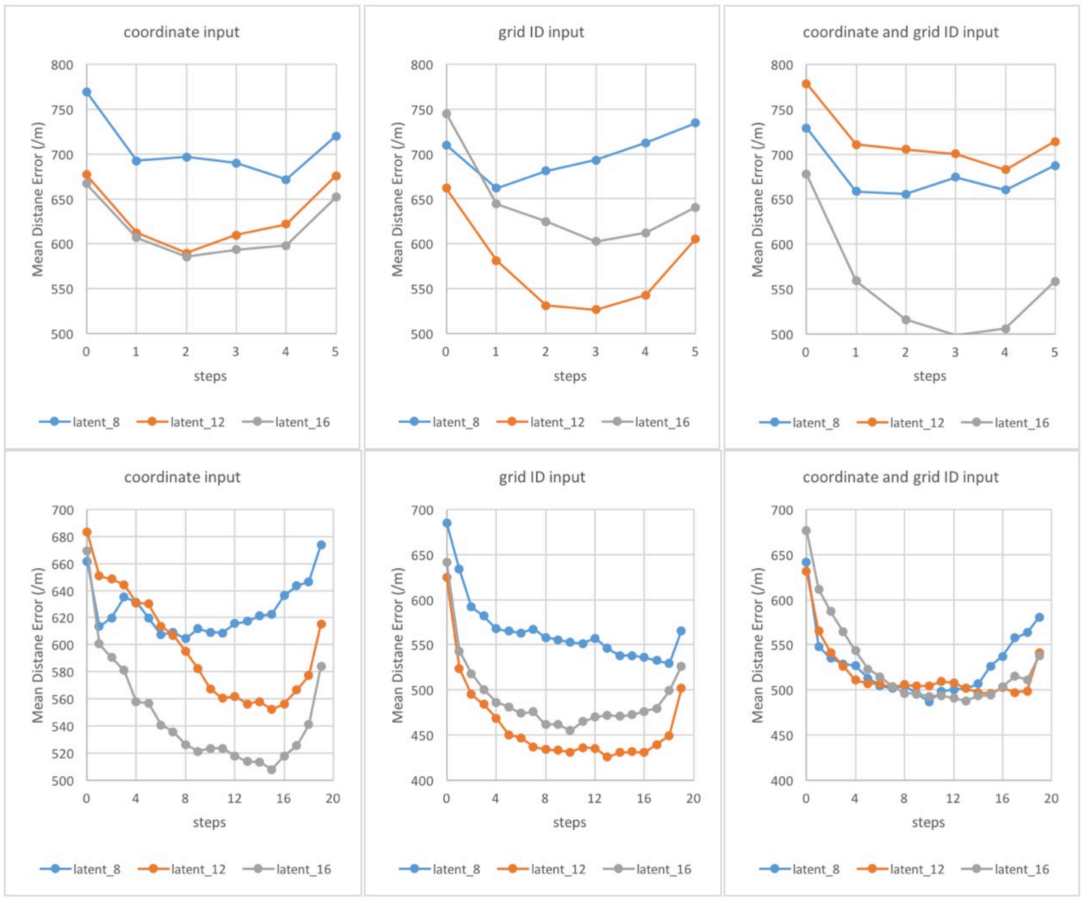


FIGURE 4.10: comparison between raw data and processed data.

reasonable as the figure shows. Moreover, when we check the patterns of generated trajectories, it looks reasonable from visualization.

4.3.2 Evaluation

The results of SVAE is shown in previous section. We give a quantitatively evaluation of our results in this section.

Owing to the lackness of exited generative model for trajectories of human mobility, we evaluate our results just using the designed Mean Distance Error (MDE).

$$E_j = \frac{\sum_{i=1}^{N} dis(l_ij, \hat{l}_ij)}{N}$$

where $dis(a, b)$ calculate the distance of point a and point b using their coordinate values; l_ij is the groundtruth, and $\hat{l}_ij$ is the outputs of the SVAE model.

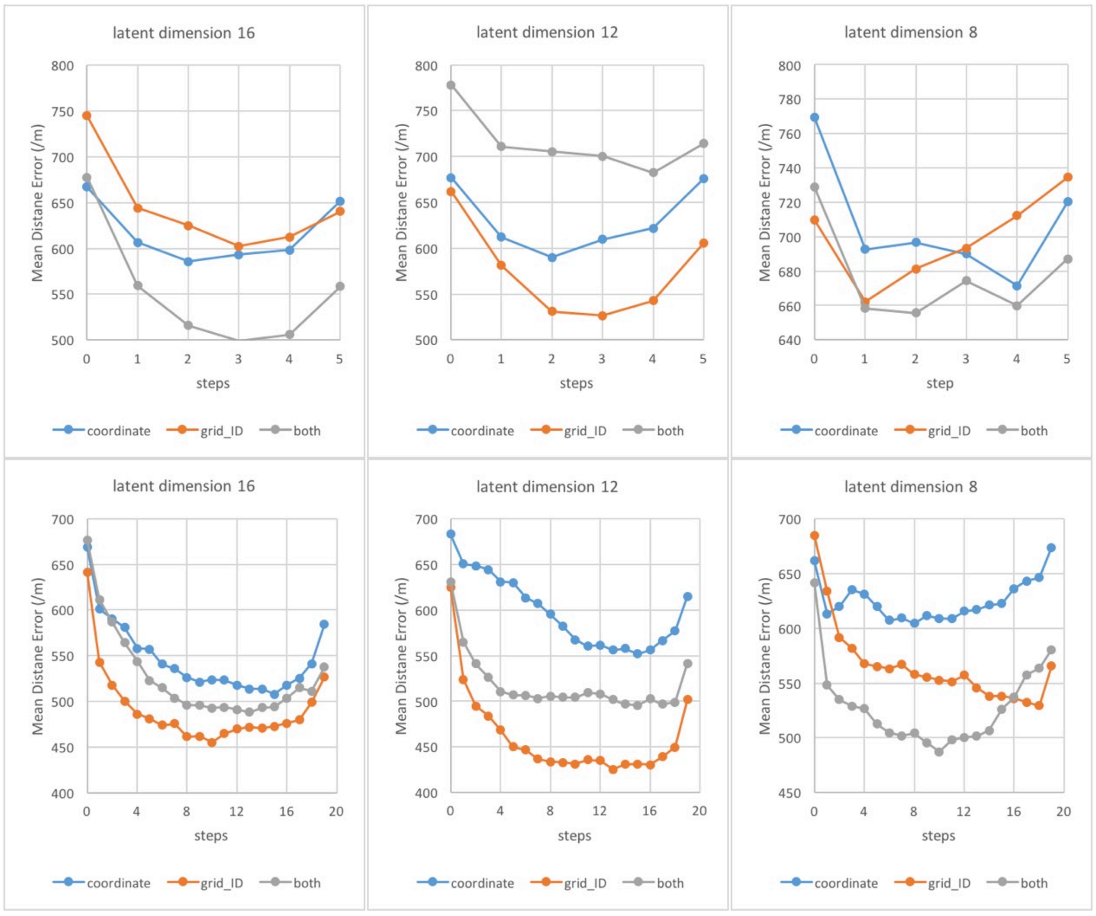


FIGURE 4.11: comparison between raw data and processed data.

In figure 4.10, we aim to give a brief comparison of the performance of the SVAE model of different dimensionality of latent space in different parameter settings. The first row is the MDE of short sequences using different inputs, and the second row is the MDE of long sequences.

In figure 4.11, we aim to compare the performance of different input strategies. Figures in first row give a comparison between different dimensionality of latent space using short sequences, while the second row give a comparison using long sequences.

In figure 4.12, we also give a visualization of a single true trajectory chosen from groundtruth and its corresponding reconstrcuted trajectory. From the figure, we can see that the driver moves from south-east to north-west in about 20 minutes. Therefore, the locations of true record and reconstrcuted record in every 5 minutes is given to show accuracy of the results in a intuitive way.

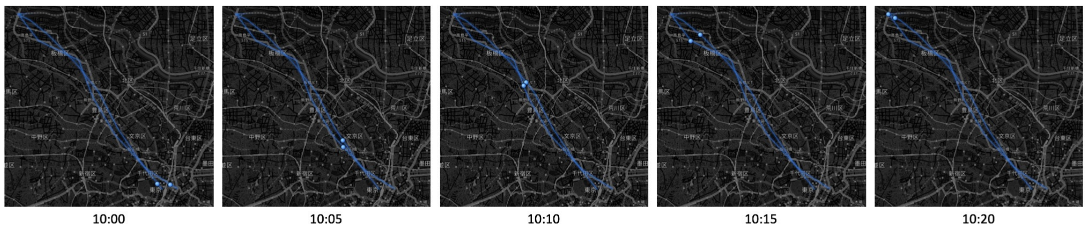


FIGURE 4.12: Comparison of singele true trajectory and fake generated trajectory.

TABLE 4.3: Reconstruction error of generated points (/m)

Longitude	1391	496	215	522	342	849	1100	341	110	134	285	312	319	335	250	66	65	92	67	48
Latitude	54	394	456	372	125	414	417	138	37	4	5	7	7	9	13	17	19	20	20	19
Distance	1392	633	504	641	364	945	1176	368	116	134	285	312	319	335	250	68	68	94	70	52

Instead of just giving a visualization of the single trajectory, a quantitative measurement is given by calcaluting the distance of two points in every step. The units of the distance error is meter. We will use this result for explaining the limitation of the SVAE model.

4.4 Discussion

4.4.1 Training performance

Two features of the training processes can be found in the figure 4.8: (1) the start points of long sequences (20) are larger than these of short sequences (6); (2) the process of minimizing loss function when just using grid ID as input is not as stable as using coordinate values or combination of both.

Besides, it should be mentioned that training the SVAE model using long sequences as inputs is more time-consuming. Therefore, The epochs of iterations should be carefully considered to reduce the computation. We set epochs of iterations to be 1,000, which make sure that the model is trained fully. By adding regularizers in our neural network layers, we can avoid overfitting.

Table 4.2 indicates three points: (1) in general, the loss values of training using both coordinate values and grid ID as input is smallest, follows training using grid ID as input and using coordinate values as input; (2) the loss values of higher dimensional latent space is often larger than those of lower dimensional latent space; (3) the loss values of long sequences is larger than those of short sequences.

A reasonable explanation of aforementioned phenomenon is that the loss function of this SVAE model is designed as the combination of reconstruction errors and Kullback-Leibler divergence of approximated posterior distribution and unit Gaussian distribution. Therefore, the phenomenon of that the loss value of training long sequences with a higher dimensional latent space is larger than others can be easily explained. Since long sequences have 20 steps, it is likely that the sum of 20 small loss is greater than the sum of 6 small loss of short sequences, which have 6 steps. That will cause the greater reconstruction error of loss function when training long sequences. The situation for higher dimensional latent space is almost the same. Higher dimensional latent space is likely to have greater values of Kullback-Leibler divergence, which is also a part of loss when training the SVAE.

From figure 4.10, we can see that models using the lowest dimensional latent space give worst performance in most cases. However, when dimensionality of latent space is setted to be 12 or 16, it is hard to say which one can lead to a better performance.

The aforementioned phenomenon could have a reasonable explanation. Since that the approximated posterior distribution constructed in latent space of SVAE model is aimed to capture as many features of input training data as possible while in a limited capacity. When the input data is very complex, then the capacity of the latent space should be larger to be able to learn the features. If the dimensionality of the latent space is limited to be small, then it will lead to the lackness of ability of learning most of the features of training data. However, when we increase the dimensionality of latent space, the ability for learning features of the SVAE model is increased indeed. But it could not be always the right strategy to increase the dimensionality of latent space since the information contained in training data is finite, which means a proper SVAE model can learn most of the salient information contained in training data using a finite dimensional latent space. Therefore, when the dimensionality of latent space is too high, the story becomes to be that part of the latent space capture the most salient features, and rest of the latent space is used to deal with the redundant trivial information. In that case, a higher dimensional latent space achieve a performance just like a lower dimensional latent space or even worse.

4.4.2 Accuracy analysis

In figure 4.11, we aim to compare the performance of different input strategies. From the figure, we can say that just using coordinate values as inputs of the SVAE model is not a good choice in most cases. However, we cannot point out the advantage of using grid ID as inputs in this figure. The comparison could be clear when we conduct more training process of the SVAE model, which is very time-consuming. Anyway, we can point out that using both coordinate vaules and grid ID as inputs can achieve a more stable results according to our daily practical experience.

Comparison pf single true trajectory and fake generated trajectory is given by figure 4.12 and table 4.3.2. According to the figure, the driver's moving in the first 20 minutes, and keep staying in the rest of time. Also, we calculate the distance error in every 3 minutes, which is shown as the table. The tendency of the distance error is reducing with time, which means that reconstructing a moving trajectory is harder than reconstructing a staying object.

In general, the results of evaluation using MDE shows that the reconstruction error of SVAE model is smaller than 800 meters. Considering that the selected experiment area is about 33,000 $m \prod$ 36,000 m, we think that the accuracy of results of the SVAE model is enough to tackle the city scale problems.

Actually, there is a trade-off between the accuracy of the reconstruction trajectories and the robustness of the ability to generate resampled trajectories. As mentioned before, the loss function of the SVAE model is consist of reconstruction error and Kullback-Leibler divergence. In practical training process, minimizing the reconstruction error will increase the accuracy of reconstructing input trajectories, while minimizing the Kullback-Leibler divergence will reduce the complexity of learned latent space.

The goal of training the SVAE model is to minimizing both reconstruction error and Kullback-Leibler divergence. However, there can be a trade-off between them as we usually add weight, smaller than 1, to one of them. When we want our model achieve higher accuracy in reconstructing trajectories, we add a small weight to

Kullback-Leibler divergence to reduce the contribution of Kullback-Leibler divergence for the whole loss. Therefore, the training process become that we care less about the complexity of the learned latent space, just make sure the output reconstructed trajectories are as accurate as possible. In that case, we can get a model of which has a very good performance of reconstructing input trajectories but a very poor performance of generate resampled trajectories. In the other hand, if we add a big weight to Kullback-Leibler divergence, we aim to train a robust model of which latent space is as simple as possible. Therefore, we are likely to get a robust model which has a poor performance of reconstructing input trajectories. Both of the aforementioned model is not the ideal model we want. In overall, as discussed above based on the evaluation and visualization of the results, we think our model is trained in a balance way.

4.4.3 Limitations

We also want to make a brief discussion about the limitation of the current SVAE model when handling the trajectories of human mobility. As shown in figure 4.12, a real trajectory and its corresponding reconstructed trajectory is given. The real trajectory is tortuous while the reconstrcuted trajectory is smooth. Although the reconstruction error is small, the output reconstrcuted trajectories of the SVAE model seems to be a smooth approximation of tortuous real trajectories.

A main limitation is that many points of reconstrcuted trajectories don't located in road network. implementing map matching to the generated trajectories may solve the problem, but we believe a better choice is that change the current coordinate and grid based model to a node based model. Another idea for this problem is changing the current resampling from Gaussian distribution strategy to resampling from historical trajectories. We will describe this idea more detail in next section.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this research, we make a brief introduction about Variational Autoencoder and Sequence-to-sequence model, then combine these two frameworks to build a Sequential Variational Autoencoder. It is believed that this Sequential Variational Autoencoder is first time implemented in modeling trajectories of human mobility. We use navigation GPS data of cars in Tokyo to evaluate the performance of SVAE model. The performance of SVAE with different parameter settings and its explanation have been discussed. In general, the SVAE model can capture the salient features of input trajectories using a latent space constructed by following Gaussian distribution, then reconstruct the input trajectories. As a generative model, the ability of generating fake resampled trajectories of SVAE is also proved. Using this SVAE model, we can generate more trajectories of human mobility which have similar pattern with training data to solve the low sampling rate problem. Besides, it is a good choice for preventing the risk of privacy invasion by implementing SVAE model to learn the salient features of confidential data. Then we can reconstruct the dataset which has similar patterns but has no privacy information. In addition, this model can improve the performance of practical applications by improve the dataset on which they based.

We also note the limitation of SVAE model when implemented in trajectories of human mobility, which is that many points of reconstructed trajectories is not located in road network. A solution for that problem is improving the accuracy

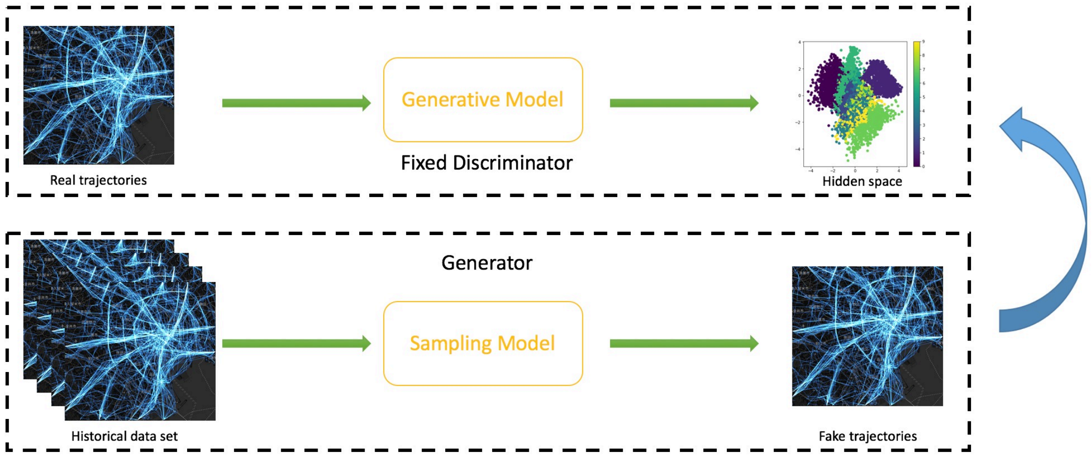


Figure 5.1: Framework of generating trajectories from historical data.

of the reconstructed trajectories which is impossible since that will reduce the robustness of the SVAE model.

5.2 Future Work

For the purpose of solving the limitation of SVAE model, there could be several future works for improving the performance of SVAE model. A road network based generative model should be built, which makes sure the reconstructed trajectories can all located in the road network.

Also, we have another idea for the aforementioned problem, of which framework is shown as figure 5.1. This framework contains two parts: 1) a fixed discriminator trained using real trajectories; 2) a generator which can generate trajectories from historical data. The model should be trained by minimizing the distance of hidden spaces of real trajectories and generated trajectories. Overall, we will continue the work about generative model for human mobility.

Bibliography

[1] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):38, 2014.

[2] S PERMISSION. Generative and discriminative classifiers: Naive bayes and logistic regression. 2005.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] Jing Yuan, Yu Zheng, and Xing Xie. Discovering regions of different functions in a city using human mobility and pois. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 186–194. ACM, 2012.

[5] Yu Zheng, Furui Liu, and Hsun-Ping Hsieh. U-air: When urban air quality inference meets big data. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1436–1444. ACM, 2013.

[6] Xuan Song, Quanshi Zhang, Yoshihide Sekimoto, Teerayut Horanont, Satoshi Ueyama, and Ryosuke Shibasaki. Modeling and probabilistic reasoning of population evacuation during large-scale disaster. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM, 2013.

[7] Renhe Jiang, Xuan Song, Zipei Fan, Tianqi Xia, Quanjun Chen, Qi Chen, and Ryosuke Shibasaki. Deep roi-based modeling for urban human mobility prediction. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(1):14, 2018.

[8] Zipei Fan, Xuan Song, Ryosuke Shibasaki, and Ryutaro Adachi. Citymomentum: an online approach for crowd behavior prediction at a citywide level. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 559–569. ACM, 2015.

[9] Yu Zheng, Yanchi Liu, Jing Yuan, and Xing Xie. Urban computing with taxicabs. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 89–98. ACM, 2011.

[10] Zipei Fan, Xuan Song, Ryosuke Shibasaki, Tao Li, and Hodaka Kaneda. Citycoupling: bridging intercity human mobility. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 718–728. ACM, 2016.

[11] Xuan Song, Quanshi Zhang, Yoshihide Sekimoto, and Ryosuke Shibasaki. Intelligent system for urban emergency management during large-scale disaster. In *AAAI*, pages 458–464, 2014.

[12] Xuan Song, Quanshi Zhang, Yoshihide Sekimoto, and Ryosuke Shibasaki. Prediction of human emergency behavior and their mobility following large-scale disaster. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 5–14. ACM, 2014.

[13] Xuan Song, Quanshi Zhang, Yoshihide Sekimoto, Ryosuke Shibasaki, Nicholas Jing Yuan, and Xing Xie. A simulator of human emergency mobility following disasters: Knowledge transfer from big disaster data. In *AAAI*, pages 730–736, 2015.

[14] Quanjun Chen, Xuan Song, Harutoshi Yamada, and Ryosuke Shibasaki. Learning deep representation from big and heterogeneous data for traffic accident inference. In *AAAI*, pages 338–344, 2016.

[15] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[16] Yaxiang Fan, Gongjian Wen, Deren Li, Shaohua Qiu, and Martin D Levine. Video anomaly detection and localization via gaussian mixture fully convolutional variational autoencoder. *arXiv preprint arXiv:1805.11223*, 2018.

[17] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of

images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[18] Jonas Mueller, David Gifford, and Tommi Jaakkola. Sequence to better sequence: continuous revision of combinatorial structures. In *International Conference on Machine Learning*, pages 2536–2544, 2017.

[19] Shin Asakawa. Comparison between variational autoencoder and encoder-decoder models for short conversation. 2017.

[20] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pages 3738–3746, 2016.

[21] M Ehsan Abbasnejad, Anthony Dick, and Anton van den Hengel. Infinite variational autoencoder for semi-supervised learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 781–790. IEEE, 2017.

[22] Mogeng Yin, Madeleine Sheehan, Sidney Feygin, Jean-François Paiement, and Alexei Pozdnoukhov. A generative model of urban activities from cellular data. *IEEE Transactions on Intelligent Transportation Systems*, 19(6):1682–1696, 2018.

[23] Mitra Baratchi, Nirvana Meratnia, Paul JM Havinga, Andrew K Skidmore, and Bert AKG Toxopeus. A hierarchical hidden semi-markov model for modeling mobility data. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 401–412. ACM, 2014.

[24] Kun Ouyang, Reza Shokri, David S Rosenblum, and Wenzhuo Yang. A non-parametric generative model for human trajectories.

[25] Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 2014.

[26] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *Interspeech*, pages 436–440, 2013.

[27] Zhuotun Zhu, Xinggang Wang, Song Bai, Cong Yao, and Xiang Bai. Deep learning representation using autoencoder for 3d shape retrieval. *Neurocomputing*, 204:41–50, 2016.

[28] Ian Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

[29] Charles W Fox and Stephen J Roberts. A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95, 2012.

[30] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[31] Ron J Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. Sequence-to-sequence models can directly translate foreign speech. *arXiv preprint arXiv:1703.08581*, 2017.

[32] Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. Table-to-text generation by structure-aware seq2seq learning. *arXiv preprint arXiv:1711.09724*, 2017.

[33] Sunyan Gu and Fei Lang. A chinese text corrector based on seq2seq model. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017 International Conference on*, pages 322–325. IEEE, 2017.

[34] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[35] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[36] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. *SLT*, 12(234-239):8, 2012.

[37] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[38] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[40] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[41] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[42] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

２０１８年度　修士論文　都市における人の移動性に対して VAE に基づく生成モデルの検討　黄　豆